МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «АиСД»

Тема: Иерархический список и его обработка

Студент гр. 8304 _____ Порывай П.А

Преподаватель _____ Фирсов М.А

Санкт-Петербург

2019

Цель работы.

3) Подсчитать общую длину всех плеч заданного бинарного коромысла bk. Для этого ввести рекурсивную функцию short Length (const БинКор bk).

Выполнение работы

Структура s_expr описывает груз или бинарное коромысло(tag = false, true).Чтобы не занимать дополнительную память в ней опимано объединение состоящее из груза и 2 указателей на коромысло(struct two_ptr).Через typedef на структуру заведен указатель lisp(Те при объявлении lisp bink, bink указатель на элемент типа s_expr).

Функция **lisp head(const lisp s)** возвращает указатель «головы» s_expr в случае если элемент типа s_expr не груз иначе выдает ошибку.

lisp tail(const lisp s) возвращает указатель «хвоста» s_expr если элемент типа s_expr не груз иначе выдает ошибку

lisp cons(const lisp h, const lisp t) выделяет память под новый указатель на элемент типа s_expr через оператор new (это будет коромысло в моей задаче) при этом указателем головы этого элемента становится h,хвоста t.Возвращает указатель на созданный элемент

lisp make_atom(int x) выделяет память под указатель на элемент s_expr с отрицательным тегом(груз, длина) и возвращает этот указатель

bool isAtom(const lisp s) проверяет является ли элемент грузом или длиной(tag = false), возращает значение типа bool

bool isNull(const lisp s) проверяет является ли элемент нулевым указателем , возвращает значение типа bool

read_lisp(lisp& y, std::string data) принимает ссылку на тип lisp(обрабатывается как обычная переменная типа lisp) и читает

иерархический список заданный определенным образом из строки data. На деле считывает пробелы и вызывает read_s_expr

read_s_expr(char prev, lisp& y, std::string data,int&i) в случае первого прочитанного символа(не скобки и не пробела)в теле условия else if (prev != '(') создает «грузик» или '(' вызывает read_seq

read_seq(lisp& y, std::string data, int &i) вызывает read_s_expr(), read_seq(), создается коромысло

write_lisp(const lisp x) печатает элемент на экране в случае если это атом(в нашем случае атомом может быть длина или груз), если нет вызывает write_seq(const lisp x)

write_seq(const lisp x) в случае непустого указателя рекурсивно вызывает следующую последовательность write_lisp(head(x) , out) ,write_seq(tail(x) , out).

Аналогично вводу перегруженные функции **void write_lisp(const lisp x, std::ofstream& out), void write_seq(const lisp x, std::ofstream& out)** аналогичны вышенаписанным только вводят информацию в файл

void sum(lisp p, short*s) рекурсивно проходит список и если элемент является длиной(первым элементом после '(' прибавляет его к общей переменной которая передается через указатель int *s

В **short Lenght(const lisp p)**(по заданию**)** заводится переменная total к-я передается в вызываемую здесь функцию sum.

void is_bink1(lisp&y, bool &is_bk) проверяет введенный иерархический список на соответствие бинарному дереву. Главная идея данной функции заключается в количестве элементов типа s_expr с tag == false на одном «уровне» равном 2. На каждом «уровне» часть списка проходится с головы до хвоста и если бк введено неверно на каком либо из них i !=2 и переменная is_bk = false

void destroy(lisp s) освобождает память с помощью оператора delete. Идея рекурсивной фунции состоит в том,чтобы двигаться по непустым указателям и освобождать память под них на каждом вызове «в глубину»

void create_check_write(std::string data_str, std::ofstream &fout) — для упрощения main в ней вызываются все вышенаписанные функции, вводится дополнительная проверка, которая не учитывается в is_bink1

В **main** Выводится диалоговое окно, предлагающее ввести информацию из файла или с консоли. В случае ввода из файла создается входной поток std::ifstream data.. и выходной std::ofstream fout в цикле while пока не конец файла считываем строку, вызываем create_check_write(data_str, fout);

Выводы.

Получены навыки работы со структурой данных «Иерархический список», рекурсивной обработкой её элементов.

Приложение А. Исходный код

```
#include<iostream>
#include<cstdlib>
#include<string>//getline()
#include<fstream>//ifstream()
#include<stdlib.h>
//VAR 3 Считаю сумму длин всех плеч бк
typedef int base; // базовый тип элементов (атомов)

struct s_expr;
struct  two_ptr
{
      s_expr* hd;
      s_expr* tl;
};    //end two_ptr;

struct s_expr {
      bool tag; // true: atom, false: pair
      union
      {
            base atom;//не могу инициализировать
            two_ptr pair{ nullptr, nullptr };
      } node;           //end union node
};                //end s_expr

typedef s_expr* lisp;

lisp head(const lisp s);
lisp tail(const lisp s);
lisp cons(const lisp h, const lisp t);
lisp make_atom(int x);
bool isAtom(const lisp s);
bool isNull(const lisp s);

void read_lisp(lisp& y, std::string data);
void read_s_expr(char prev, lisp& y, std::string data, int& i);
void read_seq(lisp& y, std::string data, int& i);

void is_bink1(lisp& y, bool& is_bk);
short Lenght(const lisp p);
```

```
void destroy(lisp s);
void sum(lisp p, short* s);
void create_check_write(std::string data_str, std::ofstream& fout);

void write_lisp(const lisp x);
void write_seq(const lisp x);
void write_lisp(const lisp x, std::ofstream& out);
void write_seq(const lisp x, std::ofstream& out);

lisp head(const lisp s)
{// PreCondition: not null (s)
      if (s != nullptr)
            if (!isAtom(s))
                  return s->node.pair.hd;
            else {

                  std::cerr << "Error: Head(atom) \n"; exit(1);
            }
      else {

            std::cerr << "Error: Head(nil) \n";
            exit(1);
      }
}

lisp tail(const lisp s)
{// PreCondition: not null (s)
      if (s != nullptr)
            if (!isAtom(s))
                  return s->node.pair.tl;
            else {
                  std::cerr << "Error: Tail(atom) \n"; exit(1);
            }
      else {
            std::cerr << "Error: Tail(nil) \n";
            exit(1);
      }
}


lisp make_atom(int x)
{
```

```
	lisp s;
	s = new s_expr;
	s->tag = true;
	s->node.atom = x;
	return s;

}

bool isNull(const lisp s)
{
	return s == nullptr;
}

bool isAtom(const lisp s)
{
	if (s == nullptr)
		return false;
	else
		return (s->tag);
}


lisp cons(const lisp h, const lisp t)
// PreCondition: not isAtom (t)
{
	lisp p;
	if (isAtom(t)) {
		std::cerr << "Error: Tail(nil) \n"; exit(1);
	}
	else {
		p = new s_expr;
		p->tag = false;
		p->node.pair.hd = h;
		p->node.pair.tl = t;
		return p;

	}
}


void destroy(lisp s)
{
```

```
	if (s->tag == false) {

		if (s->node.pair.hd != nullptr)
			destroy(s->node.pair.hd);

		if (s->node.pair.tl != nullptr)
			destroy(s->node.pair.tl);
	}

	delete s;
}


void read_lisp(lisp& y, std::string data)
{
	char x;
	int i;
	i = 0;

	do {
		x = data[i];
		i++;
	} while (x == ' ');

	read_s_expr(x, y, data, i);

} //end read_lisp
//.........................
void read_s_expr(char prev, lisp& y, std::string data, int&
i)//data[i-1] == prev
{ //prev - ранее прочитанный символ}

	if (prev != '(') {
		//data.putback(prev);
		int len = 1;
		int k = i - 1;//сохраняем положение элемента prev в data

		while (data[i] != ' ' && data[i] != '(' && data[i] != ')')
{
			i++;
			len++;
		}
```

```
            std::string str_digit = data.substr(k, len);


            int digit = atoi(str_digit.c_str());
            y = make_atom(digit);

      }
      else read_seq(y, data, i);


} //end read_s_expr

void read_seq(lisp& y, std::string data, int& i)
{
      char x;
      lisp p1, p2;
      x = data[i];
      i++;

      while (x == ' ') {
            x = data[i];
            i++;

      }


      if (x == ')')
            y = nullptr;
      else {
            read_s_expr(x, p1, data, i);
            read_seq(p2, data, i);
            y = cons(p1, p2);
      }

} //end read_seq
//.........................
// Процедура вывода списка с обрамляющими его скобками - write_lisp,
//  а без обрамляющих скобок - write_seq
void write_lisp(const lisp x)
{//пустой список выводится как ()
      if (isNull(x))
```

```
            std::cout << " ()";
      else if (isAtom(x))
            std::cout << ' ' << x->node.atom;
      else { //непустой список}
            std::cout << " (";
            write_seq(x);
            std::cout << " )";
      }
} // end write_lisp
//.........................
void write_seq(const lisp x)
{//выводит  последовательность  элементов  списка  без  обрамляющих  его
скобок
      if (!isNull(x)) {
            write_lisp(head(x));
            write_seq(tail(x));
      }
}

void write_lisp(const lisp x, std::ofstream& out)
{//пустой список выводится как ()
      if (isNull(x))
            out << " ()";
      else if (isAtom(x))
            out << ' ' << x->node.atom;
      else { //непустой список}
            out << " (";
            write_seq(x, out);
            out << " )";
      }
} // end write_lisp
//.........................
void write_seq(const lisp x, std::ofstream& out)
{//выводит  последовательность  элементов  списка  без  обрамляющих  его
скобок
      if (!isNull(x)) {
            write_lisp(head(x), out);
            write_seq(tail(x), out);
      }
}

void sum(lisp p, short* s) {
```

```
	if (isAtom(p))
		* s += p->node.atom;
	else if (p->node.pair.hd != nullptr) {

		sum(p->node.pair.hd, s);

		if (p->node.pair.tl != nullptr && !isAtom(p->node.pair.tl-
>node.pair.hd))
			sum(p->node.pair.tl, s);
	}

}

short Lenght(const lisp p) {

	short total = 0;
	sum(p, &total);

	return total;

}

void is_bink1(lisp& y, bool& is_bk) {

	int i = 1;
	lisp p = y;

	while (p->node.pair.tl != nullptr) {

		i++;
		p = p->node.pair.tl;
	}
	if (i != 2)
		is_bk = false;

	if (y->node.pair.hd->tag != true)
		is_bink1(y->node.pair.hd, is_bk);

	if (y->node.pair.tl != nullptr) {

		if (y->node.pair.tl->node.pair.hd->tag != true)
```

```
                is_bink1(y->node.pair.tl->node.pair.hd, is_bk);

        }
        else
                is_bk = false;


}

void  create_check_write(std::string  data_str,  std::ofstream  &fout)
{//создает список,проверяет список на бк,выводит результат

        lisp bin_k = nullptr;
        bool is_bk = true;
        short total = 0;


        if (data_str != "(" && data_str != ")" && data_str != "" &&
data_str != " ") {//is_bink1 не подразумевает проверку строки на эти
символы

                read_lisp(bin_k, data_str);

                if (bin_k->node.pair.hd != nullptr && bin_k->node.pair.tl !
= nullptr) {
                        if  (bin_k->node.pair.hd->tag  ==  false  &&  bin_k-
>node.pair.tl->node.pair.hd->tag  ==  false)//если  у  нач  указателя
голова или хвост грузик
                                is_bink1(bin_k, is_bk);
                        else
                                is_bk = false;
                }
                else
                        is_bk = false;
        }
        else
                is_bk = false;



        if (is_bk == true) {
```

```
            write_lisp(bin_k, fout);
            write_lisp(bin_k);
            total = Lenght(bin_k);
            std::cout << " sum = " << total << "\n";
            fout << " sum = ";
            fout << total;
            fout << "\n";

      }
      else {
            std::cout << "Бк введено неверно\n";
            fout << "Бк введено неверно\n";
      }

      if (bin_k != nullptr)
            destroy(bin_k);
}

int main(int argc,char *argv[]) {

      setlocale(LC_ALL, "Russian");
      std::cout << "Ввод из файла или из консоли? (f , c)\n";
      std::string arg;
      std::getline(std::cin, arg);

      if (arg == "f") {

            std::ifstream data(argv[1]);
            std::ofstream fout("out.txt");

            if (data) {


                  std::string data_str = "";
                  char ch;

                  while (!data.get(ch).eof()) {
                        data.putback(ch);
                        std::getline(data, data_str);

                        create_check_write(data_str, fout);
```

```
                }
        }
        else
                std::cout << "Неверное имя файла\n";
    }
    else if (arg == "c") {

        char ch;
        std::string  data_str;

        std::ofstream fout("out.txt");
        while (!std::cin.get(ch).eof()) {

                std::cin.putback(ch);
                std::getline(std::cin,data_str);

                create_check_write(data_str, fout);
        }

    }

    return 0;
}
```

ПРИЛОЖЕНИЕ В

ТЕСТЫ

Ввод

((1 2))

(1(2 3))

((2 1) 56)

((2 3)(3 4))

((2 3)(7 89))

((12((14 15)(16 17)))(22 13))

((11((12 13)(21((121 11)(111 11)))))(12((567 789)(334 112))))

((3112 1)(1 2))

(211   (2))

(

)

((5 2)((34 12)(89)))

((2 3)(45 6))

(((14 13)(12))(12 42))

((1 2))

Вывод

Бк введено неверно

Бк введено неверно

Бк введено неверно

((2 3) (3 4)) sum = 5

((2 3) (7 89)) sum = 9

((12 ((14 15) (16 17))) (22 13)) sum = 64

((11 ((12 13) (21 ((121 11) (111 11))))) (12 ((567 789) (334 112)))) sum = 1189

((3112 1) (1 2)) sum = 3113

Бк введено неверно

Бк введено неверно

Бк введено неверно

Бк введено неверно

((2 3) (45 6)) sum = 47

Бк введено неверно

Бк введено неверно

Бк введено неверно

Первые 3 строки — не проходят проверку функции is_bk1, следующие 5 строк проходят эту проверку, строки с символами (, (, «» не проходят условие if..

Фото для ((11 ((12 13) (21 ((121 11) (111 11))))) (12 ((567 789) (334 112)))) далее